2011

Prediction-based virtual instance migration for balanced workload in the cloud datacenters

Shibu Daniel

Minseok Kwon

Prediction-based Virtual Instance Migration for Balanced Workload in the Cloud Datacenters

Shibu Daniel
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623
Email: sxd5490@rit.edu

Minseok Kwon
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623
Email: jmk@cs.rit.edu

***Abstract*—Datacenters in the cloud today provide virtualized resources of CPU, memory, disk, and networks so that millions of users can use the services at the same time in an efficient and scalable way. One of the major challenges in these datacenters is load balancing and shifting. As a huge number of requests are sent to a particular datacenter or a group of servers are asked to process more than their fair share, some of the servers are overloaded, slowed down, hot spots are created, and even hardware failures may occur. This unbalanced load in the end deteriorates the performance of the entire system easily. In this paper, we propose a load balancer that aims at alleviating hot spots and distributing the load from overloaded servers to under-utilized servers. Our load balancer monitors the loads of the servers, detects indications of overloading, then migrates virtual instances from overloaded servers to target servers. We have implemented the load balancer in a real system using the Xen hypervisor. We have also conducted an event-driven simulation to evaluate the performance of our system on a large-scale. Our results indicate that our reactive-predictive load balancing algorithm helps balance load among servers in the cloud as much as the best-case scenario from the exhaustive search with much less overhead.**

***Index Terms*—datacenters, clouds, load balancing, virtual machines, migration, performance analysis, system implementation**

I. Introduction

In the 1980's, datacenters were designed primarily in support of storage [1]. Since then, datacenters have evolved significantly and today many of them offer virtualized resources in the cloud services. Datacenters are often categorized into 1) Infrastructure as a Service (IaaS), 2) Platform as a Service (PaaS), and 3) Software as a Service (SaaS). Amazon's AWS [2] has been highly successful as an IaaS, GoogleAppEngine [3] is an example of SaaS in which users can use software (e.g., Google docs) provided by the cloud, and Microsoft Azure [4] is an example of PaaS that allows users to build and run applications on the cloud platform. These clouds rely on virtualization of hardware, software and storage in order to provide their services

One of the major challenges in datacenters is load balancing and shifting. If a huge number of requests, like flashcrowds, concentrate on a particular datacenter in one geographical region, the entire system can easily be crippled. Also, if a subset of the servers need to process the majority of the work, hot spots may be created due to this unbalanced load causing elongated execution time and even sometimes hardware failures. This flashcrowds, unbalanced loads and hot spot problem is especially crucial in the cloud considering its sheer volume of work, a large number of users (a few hundred thousands [5]) and the user expectation of uninterrupted services.

To address these problems, we propose a load balancer that aims at alleviating hot spots and distributing the load from overloaded servers to under-utilized servers. Our load balancer monitors the loads of the servers, detects indications of overloading, then migrates virtual instances from overloaded servers to target servers. The overall problem is modeled as a multidimensional knapsack optimization problem [6] or bin packing [7]. Solving this problem, the load balancer predicts the future load of servers and selects the server of the lowest load as the target server for migration. The ultimate goal of the algorithm is to balance the usages of CPU, memory, disk, and networks. We have implemented the load balancer in a real system using the Xen hypervisor [8]. We have also conducted an event-driven simulation to evaluate the performance of our system on a large-scale.

Our results indicate that our reactive-predictive load balancing algorithm helps balance load among servers in the cloud as much as the best-case scenario from the exhaustive search with much less overhead. The exhaustive search examines every server regularly and moves virtual machines from overloaded servers to underloaded ones. The time for this computation is reasonably low and stays virtually at a constant. The results from our mobile-cloud prototype system also show that loads at the servers are well-balanced as the mobile client distributes work to each server disproportionately considering the current load of that server.

The rest of the paper is organized as follows. In Section II, we describe our load balancing algorithm based on the prediction model and discuss our prototype implementation with its hardware and software components. Section III discusses our initial experience with the mobile-cloud interactive system as client-directed load balancing. In Section IV, we present performance results from simulations. In Section V, we discuss related work, and conclude our work in Section VI.

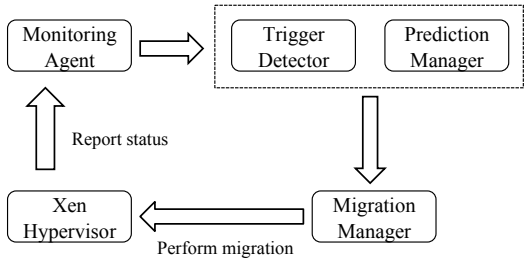

Fig. 1. Architecture of Reactive-Predictive Load Balancer

II. Reactive-Predictive Load Balancer

We design a load balancer using the prediction model based on [9], called the reactive-predictive load balancer, in order to solve the problems discussed in the previous section. Figure 1 illustrates our load balancer that consists of hardware components and software modules. The former consists of servers, storages and switches while the latter includes a monitoring agent, a trigger detector, a prediction manager, a migration manager, and a hypervisor.

The monitoring agent regularly polls the hypervisor to obtain the updated statistics of the servers. These statistics include the number of virtual instances, CPU, memory, I/O and network utilization. Independent of the monitoring agent, the trigger detector periodically scans the entire system for triggers indicating that a particular server is being overloaded, and recommends the migration of loads to other servers. When a trigger is detected, two decisions need to be made: 1) which virtual machines on the overloaded server to migrate and 2) the new destination server to migrate to. In our load balancer, the virtual machines that contribute most to the overload server is selected for migration, and the new destination server is determined via prediction whose details are discussed in the following section. The primary role of the prediction manager is to run this prediction algorithm to determine the destination server. Once the destination server is determined, the migration manager moves the target virtual machines to the new server. The hypervisor runs and manages the virtual instances.

A. Prediction and Migration Algorithm

The prediction manager views the entire set of servers as a single knapsack and solves the knapsack problem that finds the destination server. The goal here is to maximize the profit as long as the knapsack has the capacity to contain all the load. We define the profit as a function of actual utilization with respect to the thresholds for the servers. We also define the imbalance score that indicates how much the loads are not balanced in the system as follows:

$$IBScore(f,T) = \begin{cases} 0 & \text{if } f < T \\ e^{\frac{(f-T)}{T}} & \text{otherwise} \end{cases} \quad (1)$$

where f is a load utilization function for resources and T is the threshold. The goal of our prediction and migration algorithm is to minimize the overall imbalance score. The algorithm is described in Algorithm 1.

Algorithm 1 Prediction-based Load Balancing

```
1:  nodes ← Servers with utilization statistics
2:  while n ∈ nodes do
3:    if n is triggered then
4:      triggers ← triggers ∪ {n}
5:    end if
6:  end while
7:  Sort the imbalance scores of triggers in descending order.
8:  while n ∈ triggers do
9:    if attractiveness(n) > threshold then
10:     nodeToMigrate ← node
11:   end if
12: end while
13: dest ← smallestLoad(nodeToMigrate)
```

The prediction manager uses the AR family of predictors [9] to compute the attractiveness in the algorithm (line 9). The main idea is described as follows:

$$X_t = \sum_{i=1}^{p} \pi_i X_{t-i} + \epsilon_i \quad (2)$$

where X_t is the attractiveness at time t, ϵ_i is a white noise, and p denotes the order of the AR model – the higher the order, the more accurate it is. π_i is a coefficient whose value is decided based on CPU, memory, I/O and network utilization in our case. In [9], this model is applied for load balancing in the grid, and here we use the model for load balancing in the cloud.

As discussed earlier, we use the multi-dimensional knapsack problem to maximize the profit. The multi-dimensional knapsack problem [6] is an extended version of the 0-1 knapsack problem [10] formulated as follows:

$$\begin{aligned} \text{maximize} \quad & z = \sum_{j=1}^{n} p_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} w_{ij} x_j \le c_i, \quad i = 1, \ldots, m \\ & x_j \in \{0,1\}, \qquad j = 1, \ldots, n. \end{aligned} \quad (3)$$

where n is the number of items, p_j is the profit associated with item j, c_i is the capacity of item i, w_{ij} is the weight of item j consumed by resource i.

B. Hardware and Software Components

In our implementation, we use the Xen virtualization platform to create and manage virtual instances. We mainly use the Xen hypervisor to support all of the communications between virtual instances and the hardware. There are two types of the Xen domain: Dom0 and DomU. The Dom0 domain is the primary and most privileged virtual machine running on every Xen-based server. Every request for disk, memory, and networks should pass through this domain and the hypervisor to reach the underlying hardware. The DomU domain is further divided into two types: paravirtualized and hardware assisted. The paravirtualized domain is an operating system modified to run on the Xen platform like the Linux and

netware virtual machines. In contrast, the hardware-assisted virtualized domain is the operating systems whose kernel is not modified. One of the major advantages of the Xen platform is the ability to migrate domains dynamically. Xen supports three types of migrations: cold, warm, and live migrations. Refer to [8] for details.

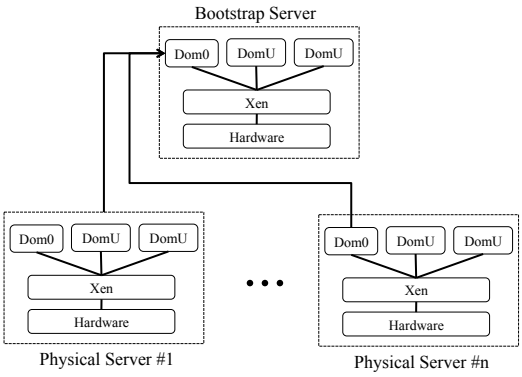

Fig. 2. Hwardware structure

We use Sun Workstations with x86, 2.2 GHz CPUs, 2GB of main memory, and two network interfaces connected to the 100 Mbps Ethernet, as depicted in Figure 2. Each computer runs the OpenSUSE 11.3 Linux operating system and Xen 4.0 on top of it. The main process, DaemonThread, runs on each computer in charge of monitoring the participating domains in communication with other DaemonThreads, hypervisors, and child processes. We also run ComputeThread and Prediction-Thread for detecting triggers, generating migration recommendations, processing predictions and scheduling migrations. For these communications, messages of the following information are transmitted through dom0's:

- Host: The IP address of dom0 stored as a string.
- dom0: This domU's IP address stored as a string.
- Id: The identifier of this message.
- CPU, mem, disk, net: The CPU, memory, disk and network usages during this monitoring round for this domU all stored as a double.

We employed *xentop* for dom0's to collect the statistics of CPU, memory, disk, and network usages. We also utilized *iostat* and *netstat* for disk and network monitoring, respectively.

III. Client-directed Load Balancing

One problem we face with our load balancing algorithm is the actual deployment to the cloud. Since it is hard to test a new load balancing scheme with the existing cloud services, one alternative approach is load balancing at the application level. The basic idea is to divide the iterative portions of the program unevenly among multiple processes (or threads) to allow some processes to compute more than others if those parts require less computation. As a proof-of-concept, we have implemented a mobile cloud system prototype that enables a smartphone (e.g., Android phone) to conduct heavy computation in interaction with the cloud. In this section, we

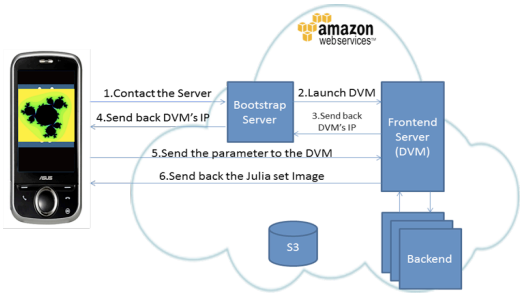

Fig. 3. Architecture of the Mobile-Cloud Prototype

discuss our initial experience with the system and how to balance its load directed by clients without any modification to the servers.

An overview of the system architecture is depicted in Figure 3. A detailed example of the overall operations is described below using the program that generates a Julia Set image in parallel computing. A client on the Android device initiates the program execution by contacting the bootstrap server (step 1). The client then receives the IP address of the DVM server from the bootstrap server (step 4). The client then establishes a connection with the DVM server and sends the command-line arguments and parameters to execute the parallel job on the backend servers (step 5). As the final step, the client receives the output of the program and displays it on the screen (step 6).

On AWS, we run three types of servers with different roles: the bootstrap server, the DVM (or frontend) server, and the backend server. We also use S3 (Simple Storage Server) to compose AMI (Amazon Machine Image) images that store these server executables. The bootstrap server is the first contact from client to server that forwards a client request to the DVM server working like a DNS server between a client and DVM. The server also starts a new instance of DVM unless an instance of DVM is already up and running. The server then fetches the IP address of the newly launched DVM and forwards it to the client device, so that all the subsequent requests from the client can be passed to DVM properly. The boostrap server itself is maintained as an AMI image, and its IP address can be maintained through either paid elastic IP addresses or a fixed IP address.

The Distributed Virtual Machine (DVM) server is an AMI where actual processing occurs. Once DVM is started, it receives command-line arguments and parameters from the client and executes the requested job in parallel using the PJC (Parallel Java) APIs. DVM runs on the frontend server, grabs the AMI of the backend servers, and launches the backend servers for parallel execution. In this case, the backend servers are ranked from 0. The outputs are gathered in the rank 0 backend server and transferred back to the frontend server. When the execution is complete, the backend server terminates if no more job is to be processed. The final outputs are sent back to the client from the frontend. Note that the client is not allowed to initiate DVM directly in order not to have many

instances of DVM whenever the client needs to process a job.

IV. Performance Evaluation

A. Experimental Setup

We evaluate the performance of our prediction-based load balancing algorithm via simulations. Our simulations create 1300 virtual instances, 400 servers, 50 switches, and 50 storage nodes. We compare three different modes of load balancing schemes:

- Best fit (bf) computes the best possible destination server for every trigger.
- Prediction (pred) computes the best possible destination server using our autoregressive predictor and usage statistics.
- Relaxed best fit (rbf) computes the best possible destination server from a subset of nodes for every trigger.

B. Results

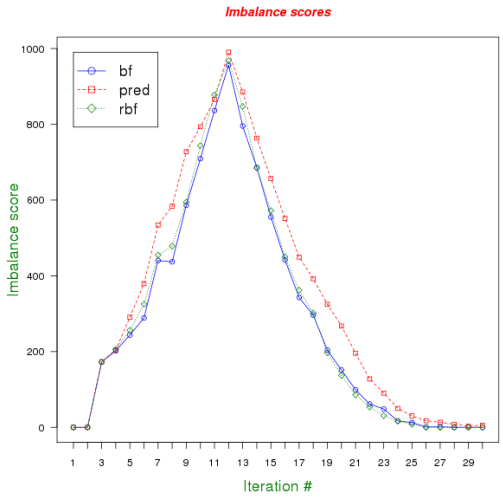

Fig. 4. Imbalance scores

Figure 4 depicts the imbalance scores (y-axis) for different iterations (x-axis). As the virtual machines join and leave at random points, the load balancing algorithm finds new destinations for overloaded servers. As the figure shows, the relaxed best fit mode performs worst because destination nodes are chosen at random not reducing the imbalance scores of the entire system–the global minimum. The best fit mode, in contrast, searches all of the servers exhaustively and works best because it selects the best possible fit. The imbalanced scores in the prediction mode increase as more virtual instances are created and decreases as those instances complete nearly close to the scores measured using the best mode. At around the 12th iteration, the load in the system peaks with 977 virtual machines active simultaneously. The prediction manager can predict this imminent high peak of overload and responds by migrating virtual machines to the servers of low load. While the results of the relaxed best fit exhibit unpredictable behavior, those in the prediction mode offer a gradual increase and decrease pattern.

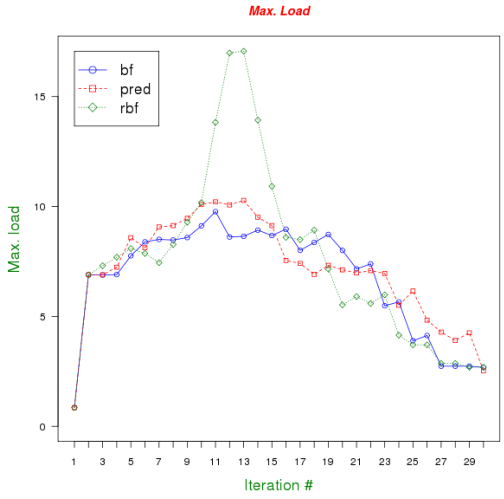

Fig. 5. Max loads

We also compare the maximum load on each server for these three different approaches. As seen in Figure 5, the maximum load of the relaxed best fit is incomparably high against the other two. The maximum load of the prediction mode is similar to the maximum load of the best fit mode. The prediction mode shows overall slightly higher maximum loads since the best fit searches the entire space while the prediction does not.

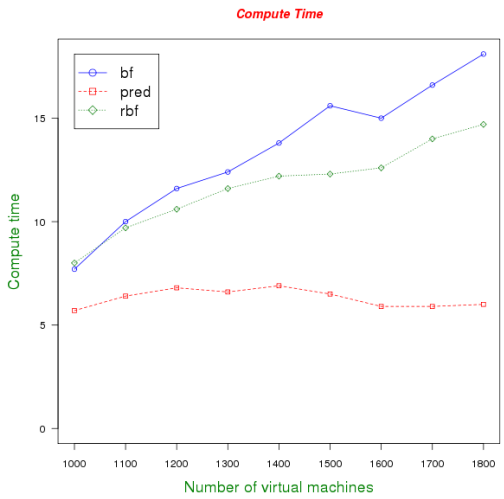

Fig. 6. Time for computation

To evaluate the complexity of each scheme, we measure the time taken to compute the solutions. In Figure 6, the x-axis denotes the number of virtual instances ranging from 1000 to 1800 and the y-axis represents the time taken to compute the target servers for migration in seconds. As the figure

shows, the time for the prediction mode stays relatively at a constant point while the other two increase almost linearly as the number of virtual instances increases. This proves that the prediction mode requires significantly low time complexity in processing migration. Although the relaxed best fit does not do exhaustive search, its time still increases since its imbalance scores should be computed and maintained like the best fit mode despite its relatively lower complexity compared to the best fit.

V. Related Work

Our algorithm is influenced by Harmony [11] in terms of the system architecture. Harmony manages hardware resources efficiently using an optimized planning component called VectorDot which balances load among virtual machines. VectorDot considers the multi-dimensionality of the resources like CPU, I/O, memory and network usage. It decides intelligently where to migrate virtual machines using load vectors and triggers. A trigger is activated when any of the resources, specifically CPU, I/O, memory or networks, cross their predetermined threshold, and a load vector stores data for current resource usages.

Rao et al. [12] designed a two-component load balancing solution to be used between datacenters. This inter-datacenter load balancer considers the prices of electricity in different datacenters and then decides to forward a client request to the most appropriate datacenter. They formulate the total cost and the goal is to minimize the total. Qin et al. [13] present weighted average load balancing with preemptive migration in which a new request is either executed on the next node in the CPU wait queue or preempts and migrates an already running job. They consider the utilization of CPU and memory to make this migration decision.

The algorithm proposed in [14] considers both the global and local computing resource usages to make load balancing decisions. Two load balancers are used: Cluster Load Balancer works at an inter-datacenter level and Local Load Balancer balances loads within a datacenter. Cluster Load Balancers form a hierarchy, share the resource usage information, redistributes the loads if needed. Wood et al. in [9] designed Sandpiper that employs two monitoring strategies–black box and gray box. In the black box strategy, the system gathers statistics from dom0's which obtains the data from the physical servers; in the gray box, the system acquires data only from domU's. The system then detects a hot spot using the gathered information and the service level agreement.

In [15], an autoregressive model is used to predict computational loads in grid computing. Bohra et al. [16] apply power modeling to the virtual cloud called VMeter. The system again monitors the load, builds the profiles, and then computes the power of virtual instances for migration.

VI. Conclusions

We have investigated the load balancing and virtual machine migration problem among servers in the cloud. Overloaded servers and hot spots in the cloud may deteriorate the performance significantly and cause even hardware failures. One natural response to the overload problem is to migrate virtual instances from an over-utilized server to an under-utilized server. In this paper, we have proposed an efficient load balancing algorithm using the prediction model. Our system collects the statistics of servers, detects a trigger for server overload, then moves some of the virtual machines from the overloaded server to the target server. To determine which virtual machines to migrate and the target server, we use the prediction algorithm based on the knapsack problem. We have implemented our system using the Xen hypervisor as a proof-of-concept, and have simulated to evaluate whether loads are indeed balanced, the overheads of migration, and the overall time. Our results show that our algorithm that computes imbalance scores using the prediction model achieve load balancing while not requiring high overhead compared to the exhaustive search mechanism. Motivated by difficulty in the deployment in the real cloud, we have developed a mobile cloud system in which we can balance load by assigning different iterations to threads at the application level.

VII. Acknowledgments

The authors would like to thank Yash Sachde, Amruta Shah, Anupam Saini for their help in developing our mobile-cloud prototype.

References

[1] Wikipedia, "Data center," http://en.wikipedia.org/wiki/Data_center.
[2] Amazon, "Amazon Web Services (AWS)," http://aws.amazon.com.
[3] Google, "Google App Engine," http://code.google.com/appengine.
[4] Microsoft, "Windows Azure," http://www.microsoft.com/windowazure.
[5] TechCrunch, "," January 2008, http://techcrunch.com/2008/04/21/who-are-the-biggest-users-of-amazon-web-services-its-not-startups/.
[6] J. Puchinger, G. Raidl, and U. Pferschy, "The Core Concept for the Multidimensional Knapsack Problem," in *Proceedings of Evolutionary Computation in Combinatorial Optimization*, February 2006.
[7] J. Bentley, D. Johnson, F. Leighton, and C. McGeoch, "An Experimental Study of Bin Packing," in *Proceedings of the Annual Allerton Conference on Communication, Control, and Computing*, 1983, pp. 51–60.
[8] Novell, "Xen Virtualization Architecture," http://www.novell.com/solutions/virtualization-workload/virtualization.html.
[9] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and Gray-box Strategies for Virtual Machine Migration," in *Proceedings of USENIX NSDI*, May 2007.
[10] T. Corman, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2009.
[11] A. Singh, M. Korupolu, and D. Mohapatra, "Server-Storage Virtualization: Integration and Load Balancing in Data Centers," in *Proceedings of ACM/IEEE Conference on Supercomputing*, November 2008.
[12] L. Rao, X. Liu, M. Ilic, and J. Liu, "MEC-IDC: Joint Load Balancing and Power Control for Distributed Internet Data Centers," in *Proceedings of ACM/IEEE ICCPS*, April 2010.
[13] X. Qin, H. Jiang, Y. Zhu, and D. Swanson, "Boosting Performance of I/O-intensive Workload by Preemptive Job Migrations in a Cluster System," in *Proceedings of USENIX NSDI*, November 2003.
[14] A. Boukerche and R. E. de Grande, "Dynamic Load Balancing Using Grid Services for HLA-Based Simulations on Large-Scale Distributed Systems," in *Proceedings of IEEE/ACM DS-RT*, October 2009.
[15] Y. Wu, Y. Yuan, G. Yang, and W. Zheng, "Load Prediction Using Hybrid Model for Computational Grid," in *Proceedings of IEEE/ACM International Conference on Grid Computing*, September 2007.
[16] A. Bohra and V. Chaudhary, "VMeter: Power Modelling for Virtualized Clouds," in *Proceedings of IEEE IPDPSW*, April 2010.